Midori Overview

2013

Midori

Safe concurrent applications that interoperate with Windows

Carry Microsoft across disruptive HW trends

- Multicore, heterogeneous processors
- Intel, ARM

Tackling "tough" problems

- Security, correctness
- Pervasive blocking, interactivity and responsiveness

Opportunistically looking at evolving legacy

- Application model
- State separation to enable cloud and roaming

Leveraging the success of Windows

- Desktop integration, host Windows apps
- Loosely coupled systems for data center scenarios

How We Work

Incubation driving to ship quality with a strong "engineering culture"

- Design and code reviews
- Automated unit tests for everything
- Test coverage, several components 100%
- Hygiene (PREFix, FXCop, etc.) and Performance Gates
- Lab available both on-demand and scheduled

Team is all developers (no PMs or SDETs)

- "Everyone codes; everyone loves to code"

Open architecture and code

- Code review email widely circulated; feedback encouraged
- All developers do level appropriate architecture, coding, testing
- Friday Morning Design Reviews

Apply the same culture for online services as the OS itself

We're hiring!

Midori OS Architecture

Runs in HyperV or direct HW (x86, x64, ARM)

- Small, policy-free microkernel (C++) for HW interaction
- Domain kernel (C#) contains most policy: schedulers, memory, resource management

"No lies", no virtualization of resources (paging)

Type safety, capability-based security

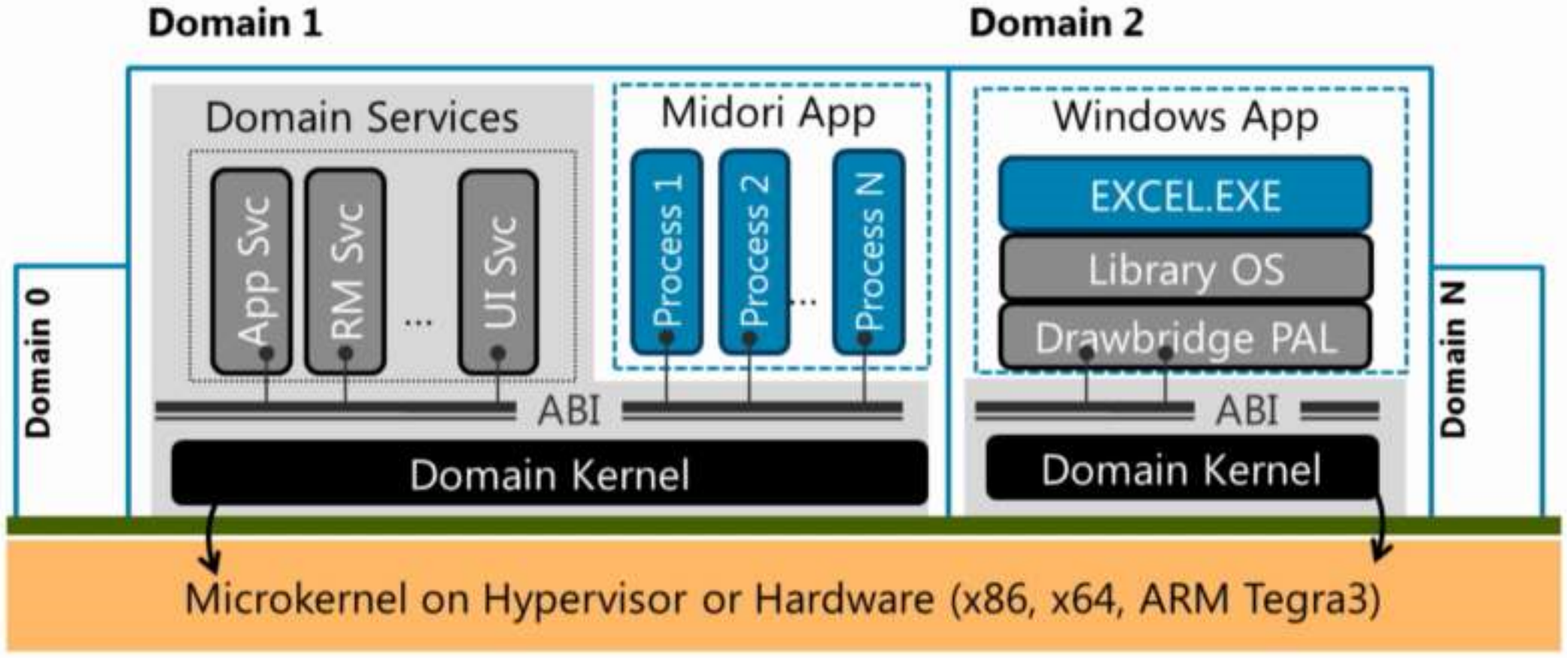
- Most of the system
- All 3rd party code, including device drivers as processes

Isolation in depth

- Software isolation between processes (type-safety)
- Hardware isolation between domains for untrusted code

Drawbridge for Windows compatibility

- Host Windows apps in a "Library OS"
- Drawbridge PAL transforms syscalls into Midori ABIs



Programming Model

Concurrency for Everyone

"A programmer of ***ordinary skill*** writes code in a ***natural*** manner and achieves results that are ***safe and scalable*** in a ***fraction of the time*** compared to state-of-the-art concurrency models."

Development Platform

Safety and Correctness By-Construction

- Correct, Secure
- Responsive, High Performance, Scalable

Midori C# and Aggressive Static Analysis

- If it compiles, it's safe – even device drivers
- No mutable statics, capability-based security
- Immutability, isolation, modern error model (contracts)

C# Productivity with C++ Performance

- Eliminates whole classes of errors (corruptions, leaks)
- Allocation-free approaches to common problems (e.g., sub-arrays, strings)

Safe Concurrency

- Lightweight processes (concurrency "for free")
- No locks or threads: no race conditions, deadlocks, priority inversions

Best in Class Developer Usability and Productivity

- Fast builds, clear separation between "App" and "OS"
- Midori C# and Midori Development Kit (MDK) fully integrated with Visual Studio

Device Driver Model

Each driver isolated in its own process

- No third party code in Ring 0
- All third party code is verifiably safe
- Each driver runs as a process – failure is isolated

Driver model based on a rich framework

- Same programming models as regular applications, no separate SDK and DDK
- Hardware resources are abstracted
- Framework largely declarative; handles queuing and resource management
  - SATA driver is 60% smaller than Windows equivalent
  - Intel 82598/Linux: 29,163 lines; Intel 82599/Midori: 2,356 lines

Driver support is complete enough to run on common hardware

- 10GB NIC, SATA, USB
- Devices: Tegra3, Dell 5x00 desktops, Bing server SKUs, X1 Carbon Touch laptop

Midori OS Architecture

Runs in HyperV or direct HW (x86, x64, ARM)

- Small, policy-free microkernel (C++) for HW interaction
- Domain kernel (C#) contains most policy: schedulers, memory, resource management

"No lies", no virtualization of resources (paging)

Type safety, capability-based security

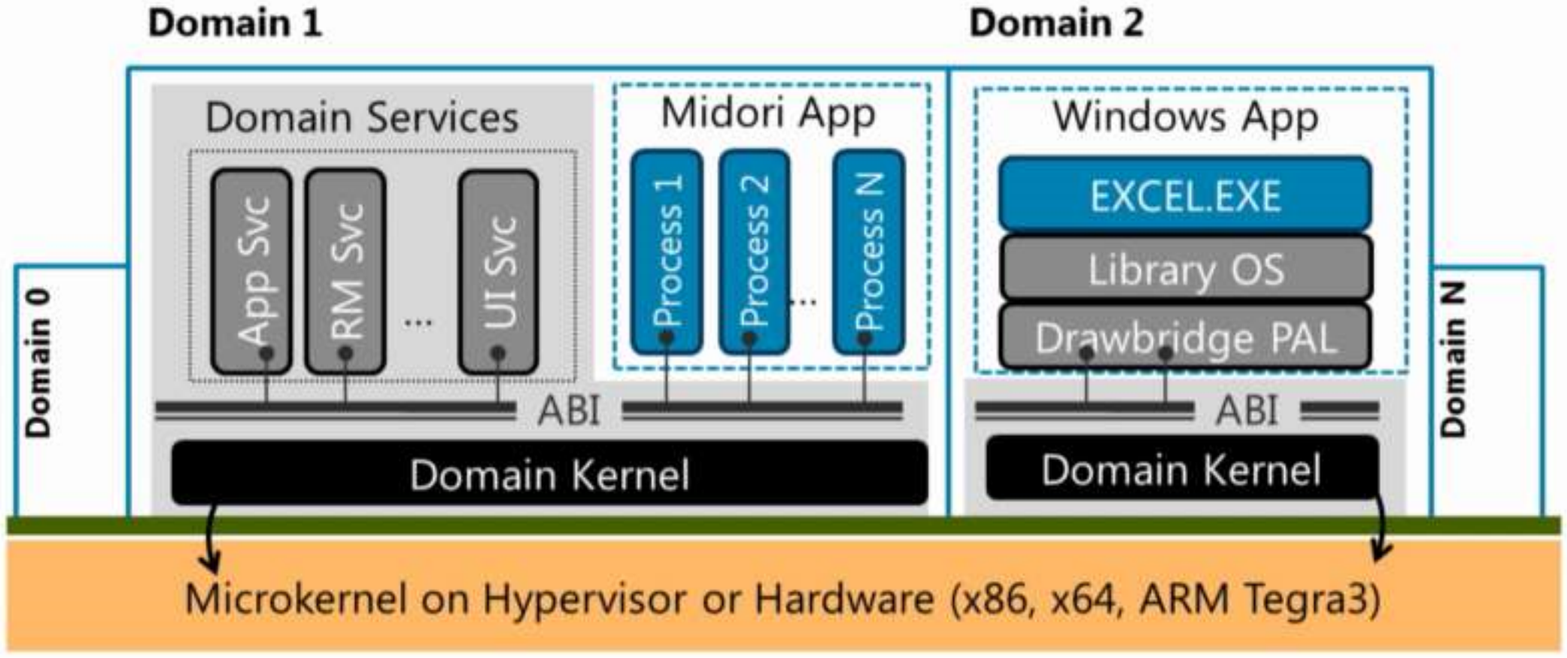
- Most of the system
- All 3rd party code, including device drivers as processes

Isolation in depth

- Software isolation between processes (type-safety)
- Hardware isolation between domains for untrusted code

Drawbridge for Windows compatibility

- Host Windows apps in a "Library OS"
- Drawbridge PAL transforms syscalls into Midori ABIs



Device Driver Model

Each driver isolated in its own process

- No third party code in Ring 0
- All third party code is verifiably safe
- Each driver runs as a process – failure is isolated

Driver model based on a rich framework

- Same programming models as regular applications, no separate SDK and DDK
- Hardware resources are abstracted
- Framework largely declarative; handles queuing and resource management
  - SATA driver is 60% smaller than Windows equivalent
  - Intel 82598/Linux: 29,163 lines; Intel 82599/Midori: 2,356 lines

Driver support is complete enough to run on common hardware

- 10GB NIC, SATA, USB
- Devices: Tegra3, Dell 5x00 desktops, Bing server SKUs, X1 Carbon Touch laptop

Networking

Complete client and server-side stack

- TCP, HTTP, UDP
- HTTPS on the client and server
- Inter-process data transfer uses immutable data buffers
- Uses formalism (Hierarchical State Machines) to manage complex concurrency

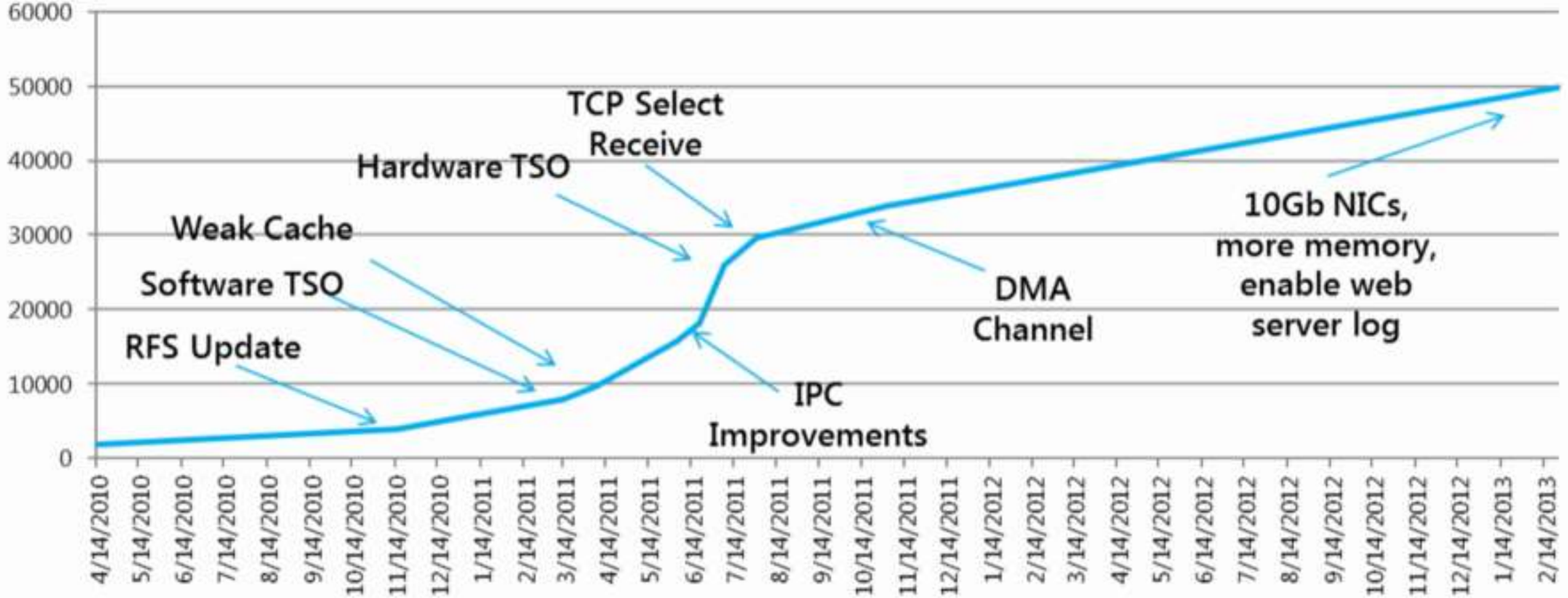
Achieves performance requirements without sacrificing security

- Separates the data and control paths
- Inter-process data transfer uses immutable data buffers
  - End to end "zero copy"
  - Buffers cacheable throughout the system, without risk
- DMA Channel for efficient data transfer between applications, driver, and hardware
- Can drive 10Gb NIC at wire speed using a single CPU core
  - Simpler architecture and more than 3x efficiency than Windows.

Network Security

- Extensive fuzzing of various network protocol implementations on Midori
- Not a single vulnerability that enables remote execution, validation of Midori's safety approach

Case Study: SPECWeb99 Historical Data



Current Hardware:
2x10Gb NIC, SSD, 8 core CPU, 48GB memory

Midori Development Kit (MDK)

Developer experience is Visual Studio based

- Reuse VS project system, as much of the IDE as possible
- Some customization for build system and new tools (async debugging)

Midori extends C# with systems and safety features

- Modern error model – exceptions and contracts
- Isolation, immutability, and controlled side-effects
- Lifetime analysis and stack allocation
- 1st class support in Visual Studio: IntelliSense, statement completion

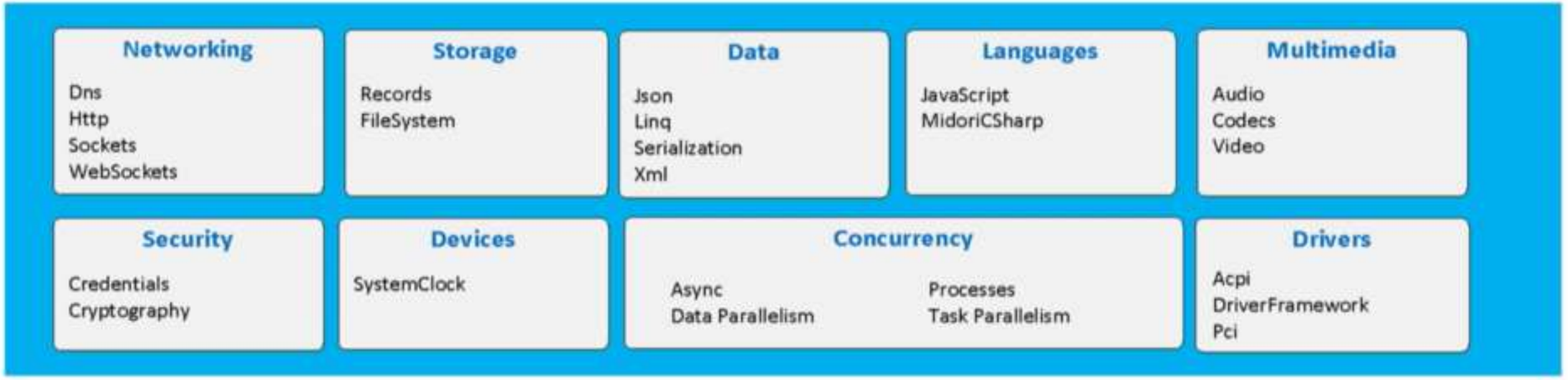
System is packaged as a set of reference assemblies

- Model is similar to .NET / Windows 8
- Rich Framework, V1 targets Server and Cloud; Client soon afterwards

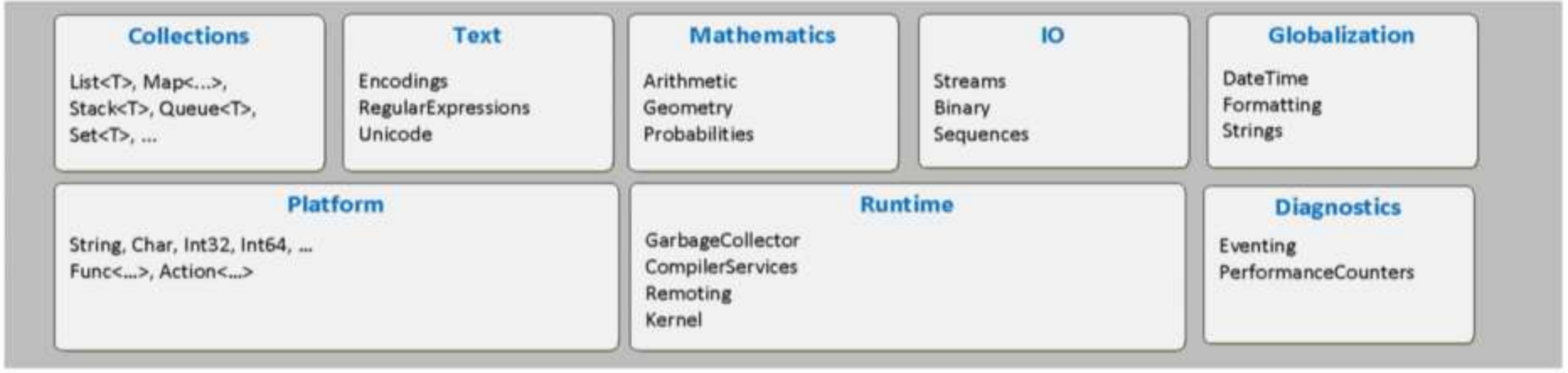
10

PROGRAM MODELS

FRAMEWORK



BASE CLASS LIBRARIES



Concurrency Model

Process Parallelism

```
SpeechEngine engine = factory.CreateProcess<SpeechEngine>(…);
SpeechRecoResult reco = await engine.Recognize(…);
```

Lightweight, isolated processes

"Blocking" coroutines

Task Parallelism

```
immutable LanguageModel model = … ;
isolated List<Phoneme> phonemes = …;
AsyncResult<RecoScore> scoreTask =
    async.Run([consume phonemes] () => {
        engine.Score(model, phonemes);
    });
  …
RecoScore finalScore = await scoreTask;
```

Spawn a task, no unsafe state capture

Ownership transfer of mutable state

Safe use of immutable data

Data Parallelism (Implicit, Libraries, Declarative Models)

```
// Example LINQ queries with side effects are banned
var myQuery = from x in xs where x.f++ select x;
```

Race condition rejected at compile-time

Deep Immutability in Frameworks

```
public immutable class MyComparator : AComparator<My> { … }
```

Scheduling and Resource Management

Kernel, User Mode Schedulers

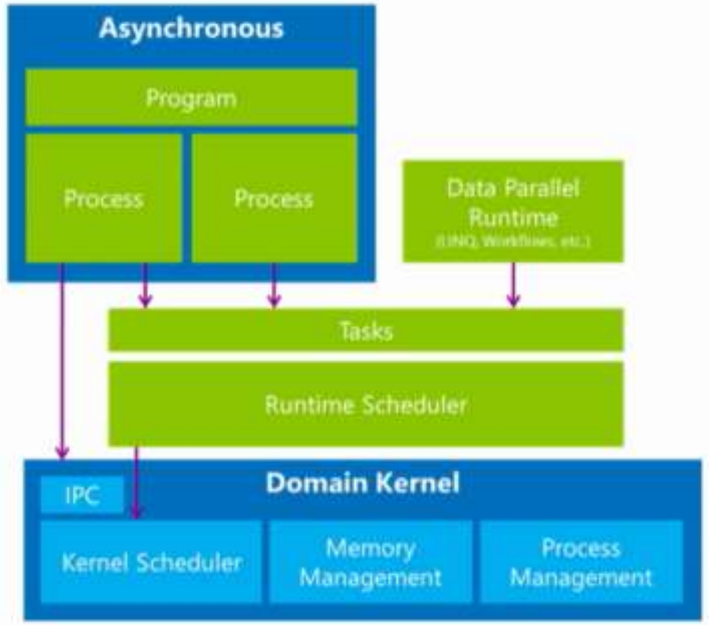
- Message passing, task and data parallelism
- Cooperative scheduling: less preemption, long quanta, warm caches
- Unified interrupt dispatch pathway

"Non-blocking" user code

- Linked stacks and small coroutines
- Begin life with a small stack, adaptively grow as needed
- Mostly C# **await** model – but efficient waiting, no state machines

System-Wide Visibility

- Mechanism is embedded into the platform
- Policy evaluated at multiple levels, all the way up to application code
- Not expressed using "priorities"
- System understands all interactions between processes and domains



Case Study: Lync

Windows

Applications explicitly manage the threads and the kernel scheduler

Some threads are set manually to real-time values!

Lync

- More than 90 threads of varying priorities (Some at priority 21!)

Hard to get right

Breaks down under stress

Interacts with other programs poorly

Midori

No threads for applications to manage

Resource management is done declaratively

Midori Lync

- Uses 17 processes, all sharing a common scheduler reservation; the resource manager handles the rest

Simple to get right

Stable under stress

Interacts well with other programs

Application Model

Midori programs are comprised of many small processes

- Each connected by an interface written in C#
- Standard asynchronous RPC – server stub, client proxy

A proxy to a remote object is a "capability"

- Used to send asynchronous messages
- Transferrable across processes
- Used to expose all system services
- Used for all I/O (disk, network, graphics, etc.)
- Not forgeable, all security is capability-based, no ambient principal

By construction, applications are

- Concurrent (able to do many things at once)
- Responsive (tolerant of high latency operations)
- High Throughput (able to speculate and pipeline)

Message Passing in Four Steps

1. Declare a service interface, marked [Eventual]

```
[Eventual]
public interface ISpeechEngine
{
    ISpeechResult Recognize(Stream audio);
    …
}
```

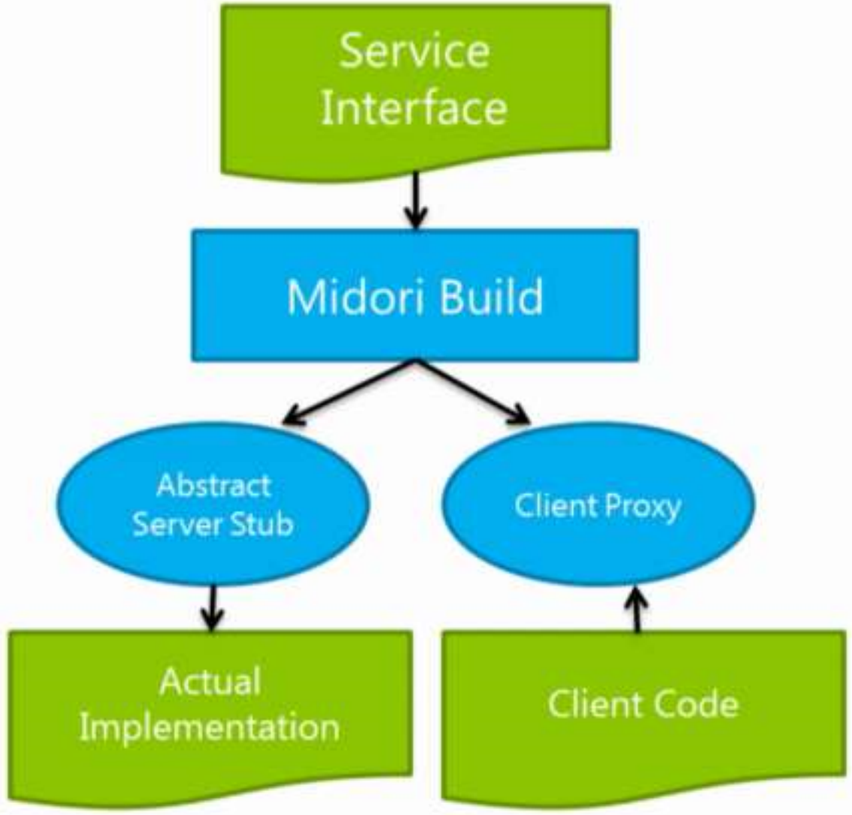
2. Compile the IDL project (or skip, if single project)

3. Implement the generated server-side stub

```
class MySpeechEngine : SpeechEngineServer
{
    public override SpeechResult Recognize(Stream audio)
    {
 …
    }
}
```

4. You're ready to use the service via its client-side proxy

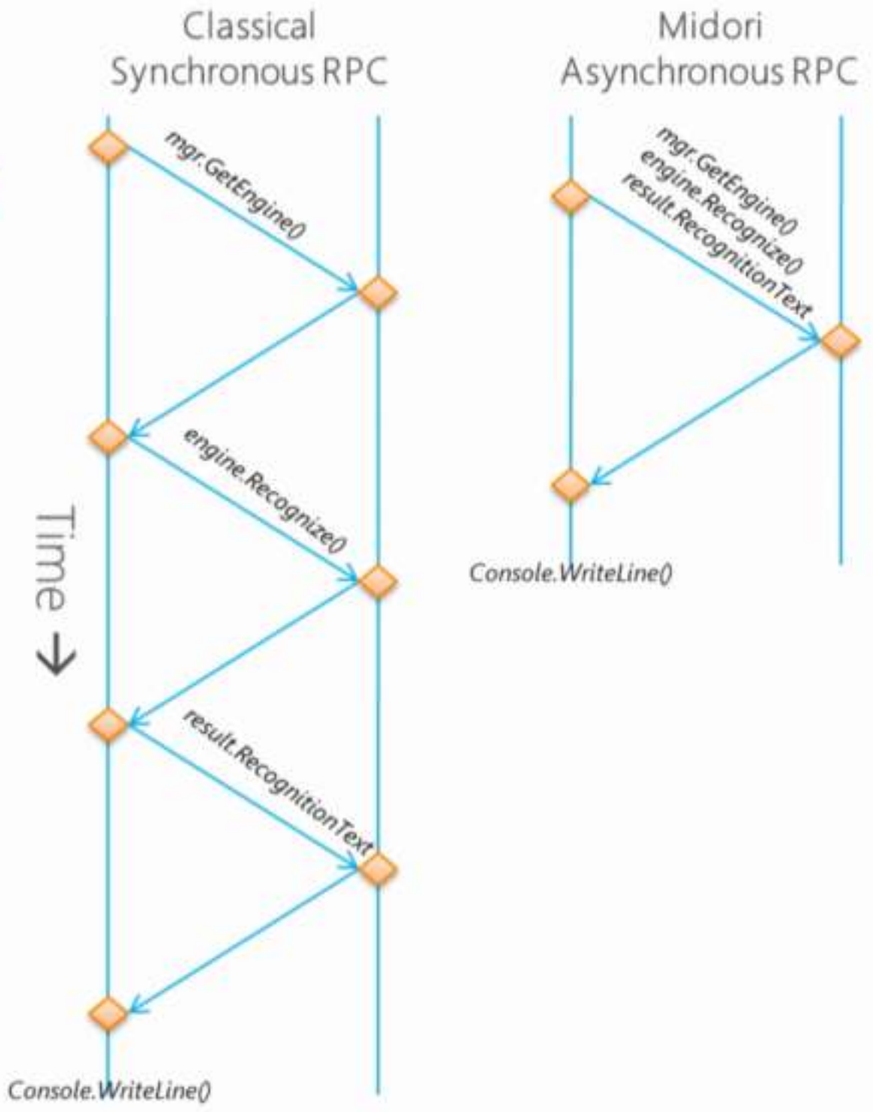
```
SpeechEngine engine = …;
SpeechResult = engine.Recognize(…);
```



Message Passing Performance

Asynchronous model masks latency

- Asynchronous masks latency
- Pipelining reduces round-trips
- Speculation absorbs latency
- Flow control improves efficiency




```
Stream audio = …;
SpeechEngine engine = mgr.GetEngine("en-US");
SpeechResult result = engine.Recognize(audio);
AsyncResult<string> text = result.RecognitionText;
Console.WriteLine(try await text);
```

Error Model

Robust error model for systems programming

- Fail fast discipline – if it's a bug, stop running code "now"

Contracts declare pre- and post-conditions

- Declarative, side-effect-free
- Compiler analysis enlightened, optimizes
- The 95% case; most places where .NET employs exceptions

Checked exceptions for "dynamic" / recoverable failures

- By default, methods may not throw
- Not possible to circumvent; allows compilers to optimize
- A method can mark itself 'throws' to opt-in
- Optionally, methods may restrict the set of exceptions thrown
  - Callers may only depend on this if annotated – no dependencies on unstated behavior

Deterministic destruction for eager resource reclamation

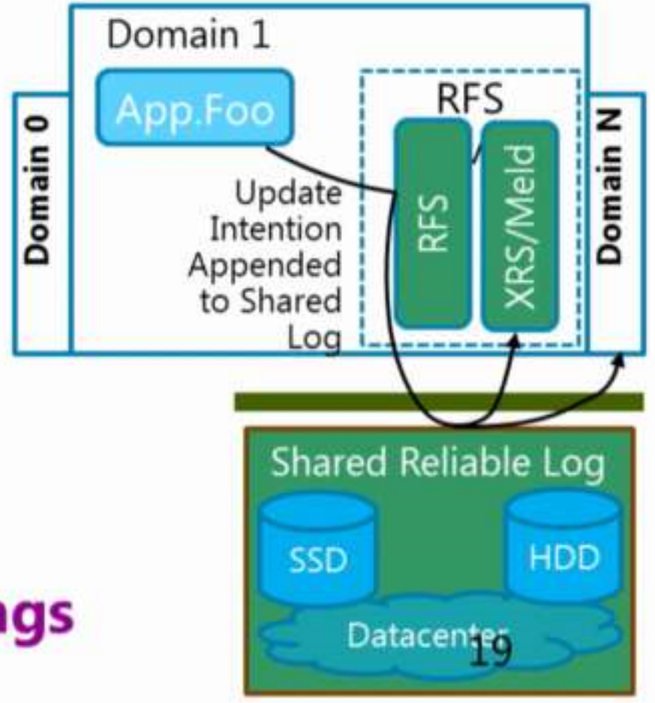
Storage

Designed for modern storage devices and application I/O patterns

- SSD, not spindle

Log Structured Transactional Record Store

- Reliability and consistency of a database
- Highly-scalable lock-free architecture
- Append-only, space reclamation (GC) based on policy



Application friendly

- Efficient snapshots and replication
- Capability-based access control model
- Declarative record-based programming model

Opportunity for data center cost and power savings

- Shared distributed logs using custom hardware

Error Model

Robust error model for systems programming

- Fail fast discipline – if it's a bug, stop running code "now"

Contracts declare pre- and post-conditions

- Declarative, side-effect-free
- Compiler analysis enlightened, optimizes
- The 95% case; most places where .NET employs exceptions

Checked exceptions for "dynamic" / recoverable failures

- By default, methods may not throw
- Not possible to circumvent; allows compilers to optimize
- A method can mark itself 'throws' to opt-in
- Optionally, methods may restrict the set of exceptions thrown
  - Callers may only depend on this if annotated – no dependencies on unstated behavior

Deterministic destruction for eager resource reclamation

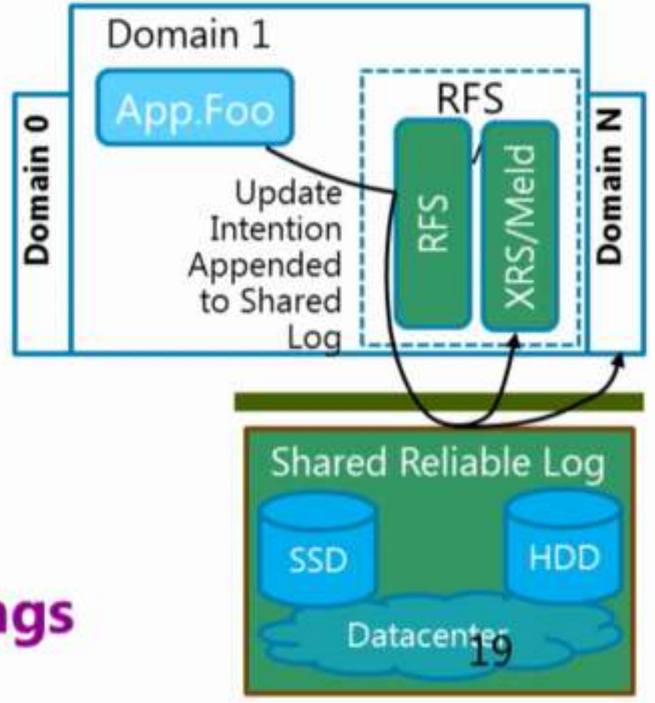
Storage

Designed for modern storage devices and application I/O patterns

- SSD, not spindle

Log Structured Transactional Record Store

- Reliability and consistency of a database
- Highly-scalable lock-free architecture
- Append-only, space reclamation (GC) based on policy



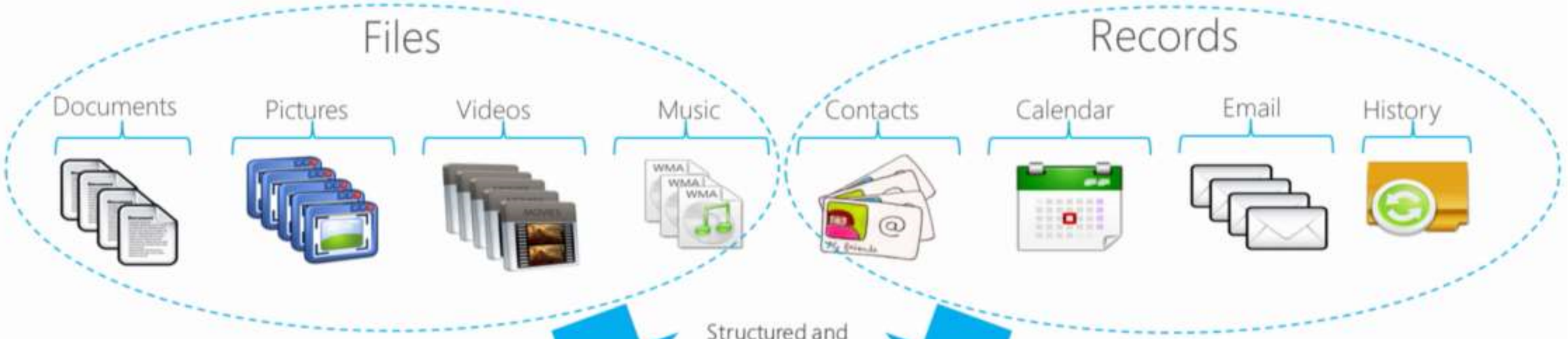
Application friendly

- Efficient snapshots and replication
- Capability-based access control model
- Declarative record-based programming model

Opportunity for data center cost and power savings

- Shared distributed logs using custom hardware

Storage Schemas and Records



Structured and unstructured data stored alike

Data schema defined in C#

Schema Provider

Transactional Record Store

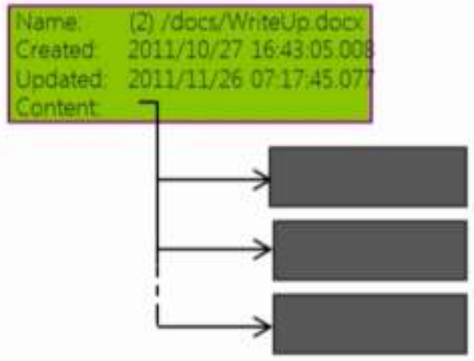
Key/Value Store
Range Operations
Multistep Transactions
Serializable Isolation

Log structured storage provides immutable snapshots to both client and cloud apps

Log

File System Schema

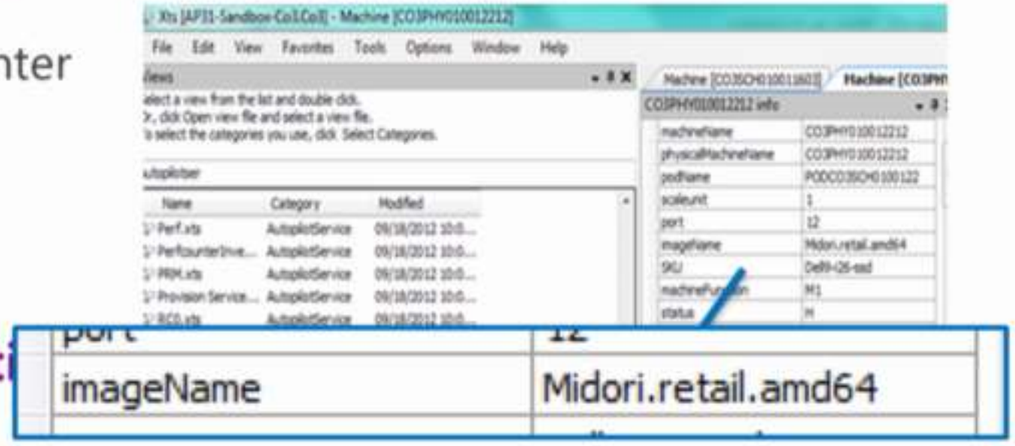
```
[Record]
public sealed partial class FileRecord
{
    //  The key is composed of Depth + FullPath (including Name)
    //  Depth is how deep the File is within the hierarchy (# of dirs above)
    //  FullName refers to the entire Path + Name of File
    //  This organization allows clustering of files within each directory
    [RecordKey(Ordinal = 0)]
    readonly ushort m_pathDepth;

    [RecordKey(Ordinal = 1)]
    readonly string m_fullName;

    ...

    [RecordContent]
    DateTime m_creationTime;

    [RecordContent]
    DateTime m_updateTime;

    //  Entire file content is stored in this field
    //  Tiny sizes (<512 bytes) are inlined
    //  Small sizes (<3K bytes) same segment
    //  Larger sizes written independently and may span segments
    [RecordContent]
    StreamSource m_content;
}
```



Cloud Data Center

Next generation Cloud support, standard SKUs, standard deployment mechanisms

- Midori is designed for the next generation data center
- Cloud-based code and data deployment
- Running on Autopilot V9, V10, and V11 SKUs
- Excellent teamwork with Autopilot team



Full integration with Bing data center automati

- Automated deployment, servicing, and reinstallation
- Midori Server looks like a Windows Server to Autopilot

Best in class debugging and monitoring

- Events and Perf Counters viewable in XTS
- Disk-based crash dumps, diagnosability is built-in

Web Programming Model

Midori Web Server

- Full HTTP support, including streaming and chunking
- Highly configurable, admission control, web apps and metabase
- Powerful object model, ASP.NET Razor also available

Rich Web Services stack

- Declarative XML and JSON serialization
- Unified programming model with message passing for program services

```
[Eventual]
[WebService]
public interface ISpeechService
{
    [WebMethod(Path="query")]
    Stream Recognize(
        [WebArgHeader(Name="Content-Type")] string contentType,
        [WebArgQuery] string locale, …
```

WebSockets coming online as we speak

Case Study: Midori Speech Server

Picked Speech Recognition as an excellent workload and partner

- Important to the company going forward (Xbox, Windows, Phone, TV)
  - Windows Phone 8X growth (2% to 15%) + increased voice drives 2X usage
  - New Xbox drives greater Kinect attach rate, immersive experience, TV, at least 5X
  - Conservatively, basic math says Speech usage will grow 5-10X in the next two years
- Challenging to meet latency and quality requirements – has hit the CPU wall

Started with existing C++ assets, architected a modern Speech Service

- Highly concurrent, Streaming rather than Batch
- 370 KLOC of unsafe C++ became 115 KLOC of safe Midori C#
- Server simplification, modernized code (collections, error model, GC)

Successfully delivered direct customer value

- Midori Speech Server now taking 20% live Windows Phone traffic; began on 12/12/2012
- Improved customer experience, 40% latency improvement
- 3-4x better data center density and capacity
- Able to deliver Deep Neural Networks (DNN), further improving quality of Speech Recognition

Windows / C++

```
HRESULT CAudioQueue::SRSiteRead(
    __out_bcount_part(cb, *pcbRead) void * pv, ULONG cb, __out ULONG * pcbRead)
{
    HRESULT hr = S_OK;

    if (m_cpInputStream && (!m_fEndOfStream)) {
        hr = m_cpInputStream->Read(pv, cb, pcbRead);

        SPAUTO_SEC_LOCK(&m_CritSec);
        if (SUCCEEDED(hr)) {
            if (*pcbRead) {
                m_ullCurSeekPos += *pcbRead;
                if (m_cClients && m_InputFormat.WaveFormatExPtr()) {
                    BYTE * pBuff = new BYTE[sizeof(CAudioBuffer) + *pcbRead];
                    if (pBuff) {
                        CAudioBuffer * pAudioBuff = new(pBuff) CAudioBuffer(*pcbRead);
                        memcpy(pAudioBuff->m_Data, pv, *pcbRead);
                        m_Queue.InsertTail(pAudioBuff);
                        m_cbTotalQueueSize += *pcbRead;
                    }
                    else {
                        hr = E_OUTOFMEMORY;
                    }
                }
                else {
                    m_ullQueueStartPos = m_ullCurSeekPos - m_ullInitialSeekPos;
                    m_ullLastRetainedAudioEventPos = m_ullQueueStartPos;
                }
            }
            if (*pcbRead < cb) {
                m_fEndOfStream = TRUE;
            }
        }
        else {
            m_fEndOfStream = TRUE;
            m_hrLastRead = hr;
        }
    }
    else {
        *pcbRead = 0;
        SPDBG_ASSERT(FALSE);
    }

    SPDBG_REPORT_ON_FAIL( hr );
    return hr;
}
```

Unsafe Arrays (Buffer Overruns)

Hidden Blocking

Locking

Manual Memory Management

Manual Streaming Implementation

Error Prone HRESULTs

Windows / C++

```
HRESULT CAudioQueue::SRSiteRead(
    __out_bcount_part(cb, *pcbRead) void * pv, ULONG cb, __out ULONG * pcbRead)
{
    HRESULT hr = S_OK;

    if (m_cpInputStream && (!m_fEndOfStream)) {
        hr = m_cpInputStream->Read(pv, cb, pcbRead);

        SPAUTO_SEC_LOCK(&m_CritSec);
        if (SUCCEEDED(hr)) {
            if (*pcbRead) {
                m_ullCurSeekPos += *pcbRead;
                if (m_cClients && m_InputFormat.WaveFormatExPtr()) {
                    BYTE * pBuff = new BYTE[sizeof(CAudioBuffer) + *pcbRead];
                    if (pBuff) {
                        CAudioBuffer * pAudioBuff = new(pBuff) CAudioBuffer(*pcbRead);
                        memcpy(pAudioBuff->m_Data, pv, *pcbRead);
                        m_Queue.InsertTail(pAudioBuff);
                        m_cbTotalQueueSize += *pcbRead;
                    }
                    else {
                        hr = E_OUTOFMEMORY;
                    }
                }
                else {
                    m_ullQueueStartPos = m_ullCurSeekPos - m_ullInitialSeekPos;
                    m_ullLastRetainedAudioEventPos = m_ullQueueStartPos;
                }
            }
            if (*pcbRead < cb) {
                m_fEndOfStream = TRUE;
            }
        }
        else {
            m_fEndOfStream = TRUE;
            m_hrLastRead = hr;
        }
    }
    else {
        *pcbRead = 0;
        SPDBG_ASSERT(FALSE);
    }

    SPDBG_REPORT_ON_FAIL( hr );
    return hr;
}
```

Unsafe Arrays (Buffer Overruns)

Hidden Blockin

Locking

Manual Memor Management

Manual Streaming Implementation

Error Prone HRESULTs

Midori / C#

```
class AudioQueue
{
    Stream m_input;

    public throws awaits int Read(in Span<byte> bytes, int readCount)
        requires readCount >= 0
        requires readCount <= bytes.Length
    {
        return try await m_input.Read(in bytes, 0, readCount);
    }
}
```

Case Study: Speech Server Performance

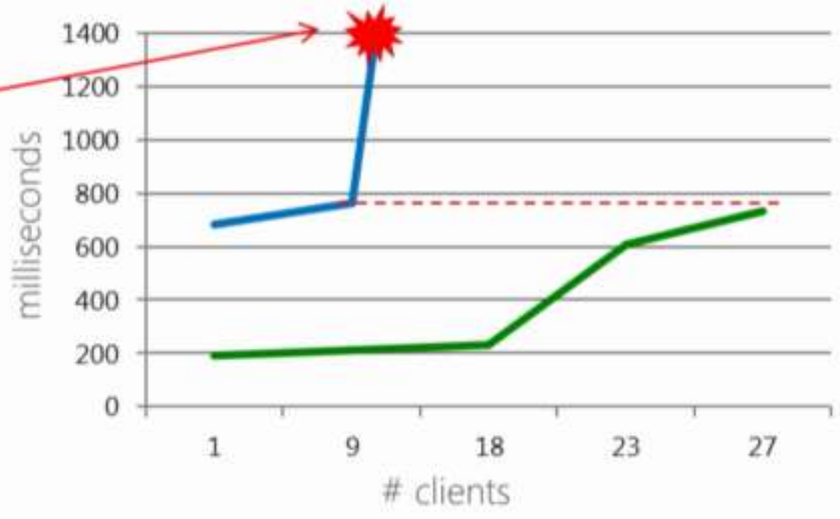
At more than triple the load, Midori delivers customer-visible latency gains

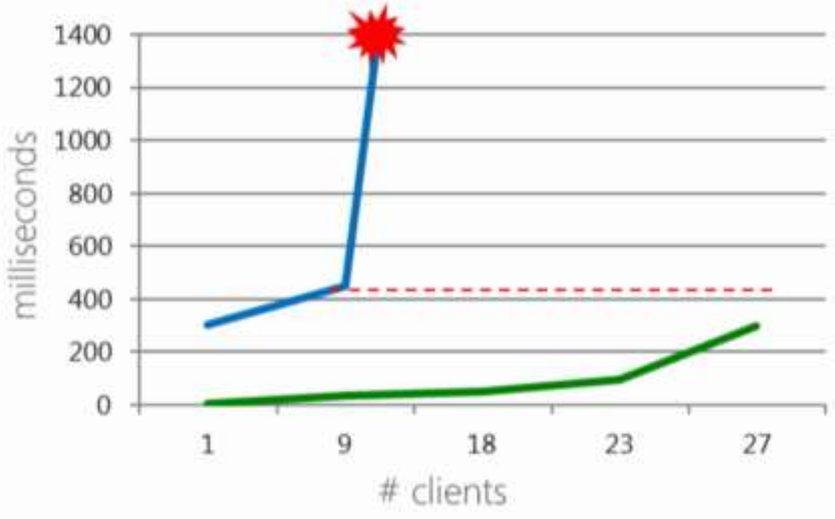
- Windows Phone beats latency target at 3x load; Xbox at 4x

	Windows Phone					Xbox				
Windows	**1**	**9**	**18**	**27**	**28**	**1**	**9**	**18**	**27**	**35**
Requests/Sec.	0.29	2.36	3.24	DNF	DNF	0.45	4.08	4.77	DNF	DNF
CPU Utilization %	8.2	55.93	100	DNF	DNF	7.13	57.23	100	DNF	DNF
Server Latency (ms)	681.96	762.31	4383.47	DNF	DNF	303.8	449.63	3473.5	DNF	DNF
Midori	**1**	**9**	**18**	**27**	**28**	**1**	**9**	**18**	**27**	**35**
Requests/Sec.	0.31	2.54	5.07	7.32	7.48	0.49	4.39	8.68	12.8	16.04
CPU Utilization %	2.98	24.58	49.51	83.72	86.69	3.99	23.72	44.11	65.78	93.72
Server Latency (ms)	193.53	213.68	234.28	606.85	735.64	5.82	32.24	51.49	94.29	297.46

Latency under load (lower is better)

Windows-based server does not finish





* Windows Speech Server cannot complete the benchmark at this concurrency level.

Case Study: Improving Speech Quality, DNN

We have successfully enabled Deep Neural Network (DNN) recognition on Midori

- MSR research to improve quality (Word Error Rate [WER])
- Industry standard for "learning and recognition", thanks to performance breakthroughs

	Count	GMM WER (Lower is better)	DNN WER (Lower is better)	Improved (Higher is better)
en-US Search	9544	29.24%	25.58%	**12.5%**
en-US Dictation	3309	21.26%	19.12%	**10.1%**

Performance on Midori is good enough to use in production – both latency and scaling

- Will flight en-US DNN in April (5% of traffic); it-IT and es-ES shortly thereafter

Lower latency

	Windows (Lower is better)	Midori (Lower is better)	Improved
Original	2.40	1.45	**60%**
DNN	2.95	0.85	**29%**
Ratio	**123%**	**58%**	-

Excellent scaling

	1 (Lower is better)	9 (Lower is better)	18 (Lower is better)
Original	193.53	213.68	234.28
DNN	121.42	138.72	183.13
Ratio	**63%**	**65%**	**78%**

Ongoing joint development effort between Midori , MSR, and IPE

- Speech engineers writing code and using Midori
- Training is still difficult; Google trains on 10X data, distributed, gets WER of 12.3%*
- Long-term partnership on distributed DNN and machine learning

* http://research.microsoft.com/apps/pubs/default.aspx?id=171498

Case Study: SPECWeb05

Speech results consistent with other workloads

Number of conforming connections (higher is better)

Workload	Windows	Midori	Ratio
Banking	34,800	69,000	198%
eCommerce	70,600	82,500	117%
Support (4 core)	27,300	47,000	172%
Weighted Average	**49,461**	**78,442**	**159%**

System

2x Quad Core CPU, 48GB RAM, 4x 512GB SSD, 2x 10Gb NIC + 2x 1Gb NIC
Logging and HTTPS enabled for both Windows and Midori runs per benchmark rules.
HyperThreading enabled.

Windows Settings

Same as official Windows submissions on SPEC.org, except for custom tunings that improves Windows performance such as affinity.
IPSEC and Firewall off

http://midori/Wiki Pages/Comparison of Midori and Windows SpecWeb05 Performance.aspx

Why Great Performance?

Midori architecture; "hands off" approach to domain-specific aspects of benchmarks

Excellent locality, immutability
- Zero-copy IO pathway (network DMA thru WebApp)
- Isolated processes, shared immutable data, excellent locality
- Compared to Windows code, no isolation, unsafe sharing of mutable state (0.582CPI vs. 0.778CPI)

Safe concurrency, no locks
- Windows code used tricky (and costly) lock-free code

Built-in scalable scheduler
- No priority inversions, longer quanta
- Windows code used manual thread pool implementation, dangerously used priorities
- Saw the same thing last year with Lync on Midori

Safe, highly optimized managed code (Bartok, Phoenix)
- No manual memory management, stack allocations (<1% GC time)
- Efficient calling conventions, aggressive inter-procedural and inter-module optimizations
- Code layout, object freezing, efficient non-blocking code

25% more on its way
- Profile guided optimizations (PGO), parallelism

Midori UI

Ride the momentum of HTML5 and Windows 8 UI

- Same story for Web Apps and Rich UI Apps
- Declarative specification, layout via CSS
- Use existing assets whenever possible (WIX, JavaScript, D2D)

Solve key issues with Windows applications

- Unsusceptible to "screen bleaching" (aka, the STA problem)
- Safe code and capability-based security
  - Impervious to UI injection attacks, escalation of privilege

Functional/Reactive core

- Formal model for how changes are made and observed
- Provides a clear, consistent timeline as the state model
- Classically suffers from size/speed penalties due to many objects, dependencies
  - Midori FRP "up-sizes" behaviors – one tree is a behavior
  - Compact array representation to save memory; no GC pointers

Midori Browser

Flagship Midori UI application

- Demonstrate the value of Midori's UI architecture
- "Reasonable subset" of HTML 5, CSS 3, and JavaScript
- IE10 JavaScript engine integrated with Midori runtime (object layout, stacks)
- Competitive performance with Microsoft's flagship browsers and runtimes

Weather Application "Port"

Application Specific Code

Lack of CSS3 Grid

Changed **88** out of **398** lines of HTML

Changed **15** out of **2,820** lines of JavaScript

Changed **6** out of **1,570** lines of CSS

Windows JavaScript Library (WinJS)

Changed **2** out of **31,284** lines of JavaScript

Changed **0** out of **4,197** lines of CSS

Drawbridge

A "virtualization" technology from MSR for hosting isolated Windows apps

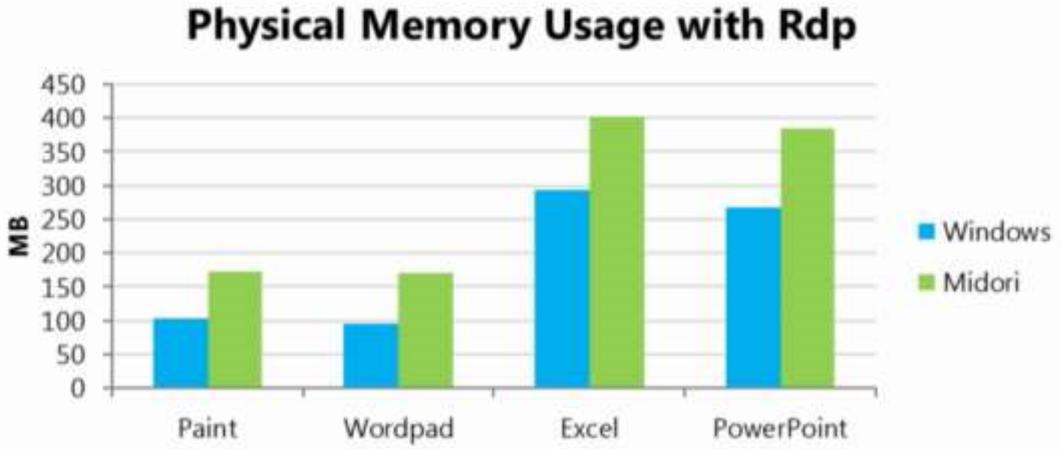
- Mostly unmodified Windows DLLs
- Host OS (Midori) implements Drawbridge ABI

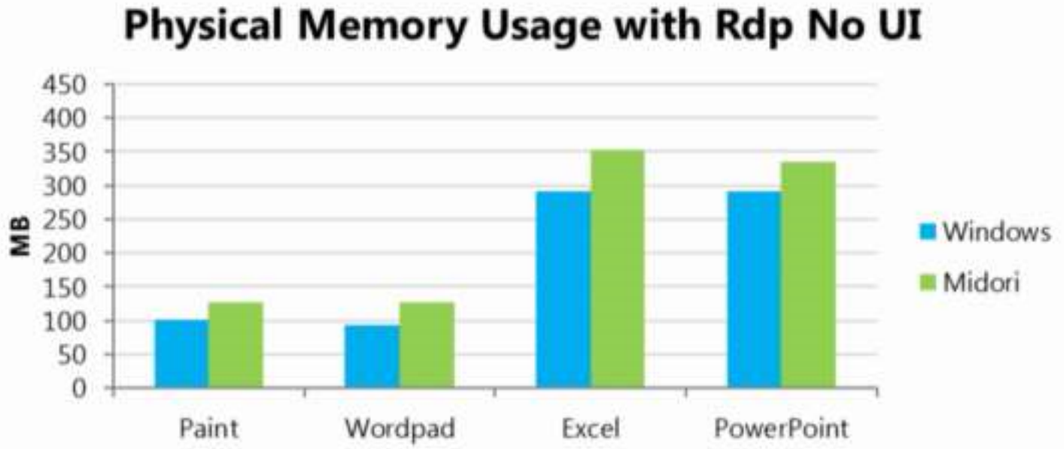
Midori's AppCompat story for rich Windows apps

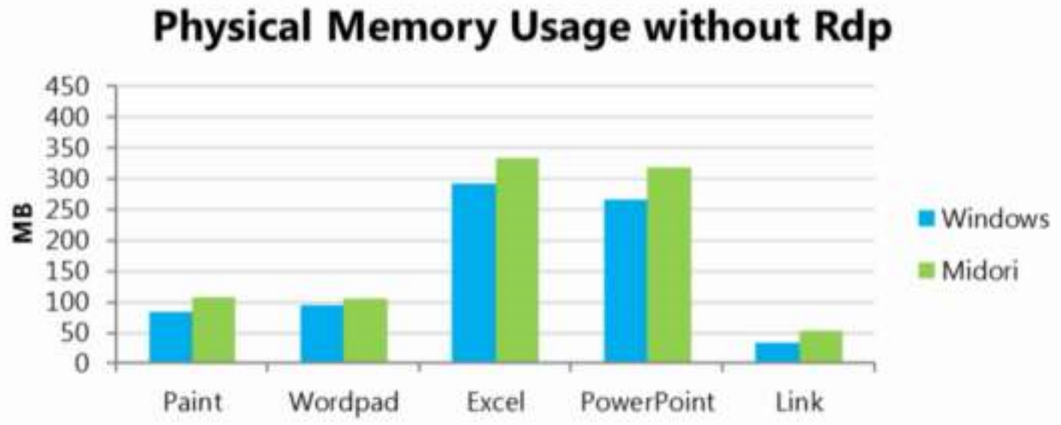
- Drawbridge PAL hosts Library OS (NT) and a Drawbridge (Windows) application on Midori
- Allows us to run Microsoft Office on Midori, for example

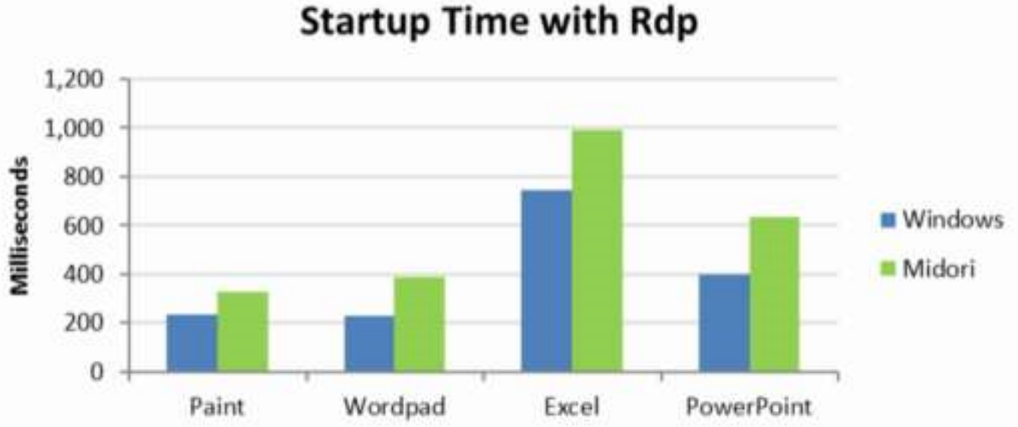
Drawbridge Performance

Midori running Drawbridge vs. Windows native









Midori Tools

Unified compiler infrastructure for C# and C++

- Ahead-of-time compilation
- Optimizer improvements accrue to both native and managed code
- Take advantage of Midori execution model
  - Example: JavaScript engine's unified object layout, linked stacks

Use existing tools (e.g. Visual Studio) whenever possible

Target many architectures

- x86, x64, ARM Tegra3, MDIL

C# Integer Benchstone x64

benchi /O2	JIT seconds (lower is better)	NGEN seconds (lower is better)	Feb-2013 seconds (lower is better)	JIT/Feb-2013 (higher is better)	NGEN/Feb-2013 (higher is better)
8queens.c	4.79	4.85	5	0.96	0.97
ackerman.c	5.33	4.65	4.53	1.18	1.03
addarra2.c	17.23	17.22	13.93	1.24	1.24
addarray.c	21.45	21.44	14.91	1.44	1.44
array1.c	8.42	8.43	7.86	1.07	1.07
array2.c	30.76	30.85	48.04	0.64	0.64
benche.c	15.6	14.79	10.64	1.47	1.39
binserch.c	10.71	10.01	7.68	1.39	1.30
bubsort.c	17.75	17.73	13.82	1.28	1.28
bubsort2.c	13.81	14.59	10.82	1.28	1.35
csieve.c	11.4	11.38	9.15	1.25	1.24
fib.c	9.34	7.81	0.01	934.00	781.00
heapsort.c	13.1	12.58	11.12	1.18	1.13
iniarray.c	27.52	27.02	2.66	10.35	10.16
logicarr.c	22.42	22.46	16.81	1.33	1.34
midpoint.c	18.46	18.49	21.02	0.88	0.88
mulmtx.c	20.65	20.66	13.42	1.54	1.54
ndhrysto.c	27.69	28.64	17.59	1.57	1.63
permutat.c	8.14	10.14	7.21	1.13	1.41
pi.c	10.95	10.95	8.6	1.27	1.27
puzzle.c	14.5	14.64	12.77	1.14	1.15
quicksrt.c	12.86	12.71	11.27	1.14	1.13
shellsrt.c	24.4	24.26	20.86	1.17	1.16
sq_mtx.c	17.67	17.75	9.47	1.87	1.87
treesort.c	3.83	3.69	3.76	1.02	0.98
tree_ins.c	9.17	9.16	9.29	0.99	0.99
xpos_mtx.c	34.04	34.01	30.13	1.13	1.13
"rw"-Geomean	**14.07**	**14.08**	**11.36**	**1.24**	**1.24**
Geomean	**13.94**	**13.83**	**8.40**	**1.66**	**1.65**

SPEC2K6 x64 /O2 vs. VS

CPU2006 INT /O2 i7	LKG3 seconds (lower is better)	Feb-2013 seconds (lower is better)	LKG3/Feb-2013 (higher is better)	LKG3 bytes (lower is better)	Feb-2013 bytes (lower is better)	Feb-2013/LKG3 (lower is better)
astar	605.5	595	1.02	93893	97440	1.04
bzip2	806.6	791.6	1.02	102222	100645	0.98
gcc	554.8	533.6	1.04	2350965	2313612	0.98
gobmk	777.2	723.7	1.07	677732	661880	0.98
h264ref	871.3	814.3	1.07	496964	514059	1.03
hmmer	484	608.9	0.79	204784	195730	0.96
libquantum	874.3	839.7	1.04	94143	92586	0.98
mcf	452.6	466.6	0.97	69112	69176	1.00
omnetpp	415.4	401.3	1.04	547996	554201	1.01
perlbench	718.1	684.9	1.05	924626	932910	1.01
sjeng	870.4	834.9	1.04	154147	149747	0.97
xalancbmk	326.4	301.3	1.08	2157540	2144857	0.99
Geomean	**615.9**	**605.8**	**1.02**	**330,628**	**328,980**	**1.00**

CPU2006 FP /O2 i7	LKG3 seconds (lower is better)	Feb-2013 seconds (lower is better)	LKG3/Feb-2013 (higher is better)	LKG3 bytes (lower is better)	Feb-2013 bytes (lower is better)	Feb-2013/LKG3 (lower is better)
dealII	515.4	522.3	0.99	573876	621763	1.08
lbm	449	453.9	0.99	87874	89282	1.02
milc	615.9	618.7	1.00	155592	164001	1.05
namd	720.2	705	1.02	375944	362612	0.96
povray	392.8	332.6	1.18	934922	919081	0.98
soplex	379.8	368	1.03	385832	374103	0.97
sphinx3	893.5	878.3	1.02	203693	197933	0.97
Geomean	**541.7**	**525.8**	**1.03**	**299,614**	**301,156**	**1.01**

Tools Improvements

Optimization

- Profile-guided optimization
- Improved runtime check elimination, driven by Speech
- SSE intrinsics for managed code
- Better interprocedural optimizations for managed code

Midori programming and execution models

- Efficient asynchronous code using linked stacks
- Frozen static objects
- Class initialization at process start-up

Developer experience

- Optimized code debugging to help diagnose production failures
- Improved compiler throughput for debug builds

Continued innovation over the next year (PGO, etc)

Summary: Great Team, Great Year

Last year, we felt good about architecture and performance

- Focus had been on architectural "hard" problems, performance
- Competitive results on industry-standard client and server benchmarks
- Demonstrated "real time" Lync on Midori, great responsiveness
- Steve & Qi asked us to tackle a real production workload

This year, Midori is delivering production Speech Recognition in Bing

- Great safety, great performance
- Immediate and very visible customer impact
- Demonstrable capacity improvement, potential for cost savings

Overall, a great validation of Midori's unique architecture

What's Next for Midori

Continuing momentum into production this coming year

- Finish V1 of the kernel and development platform
- More performance, platform innovation on its way
- Speech to 100%, geocontinental
- Deep Neural Networks and Distributed Learning
- Continue exploring partners and workloads in OSD and Azure

All design docs and code accessible to Microsoft FTEs

- http://midori/Midori Design Notes

Reminder – We are hiring!

Q&A